View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio Digital de la Ciencia y Cultura de El Salvador REDICCES

UNIVERSIDAD DON BOSCO
VICERRECTORIA DE ESTUDIOS DE POSTGRADO

TRABAJO DE GRADUACION
PROPUESTA DE DISEÑO DE UNA ARQUITECTURA DE SOFTWARE PARA COMUNICACION Y EJECUCION DE APLICACIONES REMOTAS, BASADO EN TECNOLOGIAS MICROSOFT .NET

PARA OPTAR AL GRADO DE
MAESTRO EN ARQUITECTURA DE SOFTWARE

ASESOR:
MAESTRO MARIO GERARDO CHOUSSY

PRESENTADO POR:
CARLOS EDUARDO VELASQUEZ CASTRO
JENNIFFER ELIZABETH MARTINEZ ORDOÑEZ
JUAN OTHMARO MENJIVAR ROSALES

Antiguo Cuscatlán, La Libertad, El Salvador, Centroamérica.
Septiembre de 2011

Propuesta de diseño de una Arquitectura de Software para comunicación y ejecución de aplicaciones remotas, basado en tecnologías Microsoft.NET.

Carlos E. Velásquez, Jenniffer E. Ordoñez, Juan O. Menjívar

***Resumen* — El presente propone el diseño de una arquitectura de comunicación y ejecución de aplicaciones remotas, utilizando tecnologías Microsoft .NET[1], el cual es un diseño escalable y flexible que permite adaptarse fácilmente a los cambios y nuevos requerimientos, con lo que se logra que la vida útil de la misma sea muy extensa. El proceso de desarrollo de este diseño se modela alrededor de un caso de estudio, el cual pretende recoger algunas de las necesidades más comunes en las organizaciones hoy en día. Este documento describe cada uno de los componentes que forman parte de la arquitectura, detallando la manera en la que trabajan, las librerías y funciones con las que cuenta, así como la forma en la que interactúan entre sí. Se muestran diagramas de clases UML[2] para ilustrar mejor la forma en la que operan los componentes de la arquitectura. El diseño propuesto consta de los siguientes componentes: Comunicación, Control y Ejecución, Acceso a Datos y Utilería.**

Index Terms —**Communication, components, execution, framework, proposal architecture design**

Índice de términos— **Comunicación, componentes, ejecución, plataforma, propuesta de diseño de Arquitectura.**

INTRODUCCIÓN

EL presente proyecto persigue proveer una propuesta de diseño y recomendaciones para un mecanismo de comunicación y ejecución remota de aplicaciones a través de una Red o Internet.

Más adelante en el diseño se definirán los componentes necesarios de la arquitectura, se describirán cada uno de estos componentes, sus funciones, operaciones y la manera en la que interactuarán entre sí.

Es importante mencionar que existen muchas razones para sustentar el desarrollo de una arquitectura como esta.

Para poder ejemplificar de mejor manera esta propuesta se realiza un planteamiento con base a un caso de estudio, dicho escenario establece una serie de condiciones muy comunes hoy en día dentro de las organizaciones y para las cuales esta propuesta provee una alternativa viable de solución.

En primer lugar hay que destacar que un proceso de comunicación en Red es posible fundamentarse en el uso de Sockets[3] de TCP/IP[4] ya que son componentes de uso estándar y con ello se abre la posibilidad de comunicar diferentes tipos de dispositivos y plataformas.

En segundo lugar una arquitectura basada en capas nos permite la reutilización de funcionalidades y extiende las opciones de escalabilidad en vista que es posible aislar segmentos minimizando el riesgo de afectar el resto de componentes.

[1] Versión del Microsoft .NET Framework 3.5

[2] UML: Unified Modeling Language. Herramienta para especificar o modelar estructuras, comportamientos, arquitecturas y procesos de negocio.[15]

[3] Socket, componente de software útil para realizar un intercambio de datos con otros dispositivos de red a través de una dirección de red y un puerto.[1]

[4] TCP IP según W3Schools [2] este se refiere al inglés Transmission Control Protocol e Internet Protocol y en español Protocolo de Control de Transmisión y Protocolo de Internet.

Tercero el control y conocimiento de componentes y demás herramientas tecnológicas tales como comunicación, seguridad, compresión, etc. son útiles en vista que posibilitan la toma de decisiones según las necesidades lo requieran, al mismo tiempo que es posible controlar el uso de recursos.

En vista que el propósito es presentar una guía también se definirán y detallarán las herramientas tecnológicas propuestas basadas en .NET Framework de tal modo que estas podrían utilizarse para la implementación real de esta arquitectura.

La especificación de cada capa y sus componentes se detallarán a través de un proceso de modelado de clases, definiendo nombres, características y funcionalidades específicas. Se ha escogido UML como herramienta de modelado de clases; lo anterior es debido a las bondades que UML ofrece tal es el caso que UML se considera una herramienta de modelado estándar y muy conocida, además de ser de fácil interpretación técnica y principalmente que permite modelar los comportamientos de objetos y sus relaciones.

CASO DE ESTUDIO

Consumibles S.A. es una empresa que en los últimos años ha logrado establecerse como líder en la comercialización de artículos consumibles varios para oficinas. En los últimos meses su infraestructura de herramientas de software para el manejo de pedidos y órdenes de ventas ha comenzado a generar preocupación, esto debido a limitantes respecto a conectividad, accesibilidad y lentitud en sus operaciones.

Los directivos estiman que en los próximos años la carga en el flujo de información podría duplicarse debido al rápido crecimiento experimentado, además de la inminente posibilidad del surgimiento de nuevas aplicaciones que refuercen el trabajo diario y la toma de decisiones.

El jefe del departamento de tecnología ha elaborado un estudio de dichas necesidades y ha solicitado a los Arquitectos de Sistemas elaborar una propuesta de arquitectura que solvente las necesidades, según los siguientes requerimientos:

- Una infraestructura basada en componentes que permitan la reutilización de lógica de negocios entre aplicaciones sean nuevas o existentes.
- Una infraestructura de trabajo para los desarrolladores que ayude a reutilizar componentes cuando estos sean procesos comunes entre las aplicaciones.
- Se desea aprovechar la oportunidad para que la nueva plataforma permita que los vendedores puedan conectarse a los servidores de aplicación desde diversas localidades y dispositivos sean portátiles o móviles.
- La seguridad del proceso de comunicación y el máximo aprovechamiento de recursos de red deben poder ser controladas y conocidas.
- Los protocolos Web son aparentemente muy apropiados en este caso, pero existen limitantes que se desean evitar. Entre las principales limitantes se pueden mencionar: la imposibilidad de garantizar la compatibilidad con todos los navegadores Web utilizados por los usuarios, además la existencia de aplicaciones o complementos maliciosos que puedan generar problemas en el uso de las aplicaciones corporativas. Es por eso que se preferirían aplicaciones con otro tipo de fundamento tecnológico.
- Finalmente debe permitirse una capa genérica e independiente de los demás componentes, cuyo propósito sea el acceso a diversas fuentes de datos, tales como bases de datos, archivos, Web Services[5], XML[6], etc. Esta capa podrá ir desarrollándose según se necesite en el transcurso del tiempo.

La siguiente sección presenta una propuesta de diseño que pretende dar solución a las necesidades anteriormente planteadas.

MARCO CONCEPTUAL

Para poder comprender de mejor manera el diseño de la arquitectura propuesta es necesario tener claro los conceptos, características y ventajas de las tecnologías y herramientas sugeridas, las cuales se detallan a continuación:

Como primer punto, la propuesta que se plantea basa su implementación en el .NET Framework, que

[5] Web Services Aplicacion Web que puede publicar sus funciones or mensajes al internet.[13]

[6] XML es un Lenguaje de Etiquetado Extensible muy simple, pero estricto que juega un papel fundamental en el intercambio de una gran variedad de datos.[12]

es el entorno de desarrollo de Microsoft; el cual permitirá que el software pueda ser ejecutado en diversas plataformas y dispositivos (incluyendo dispositivos móviles), siempre y cuando tengan la capacidad de soportar el entorno de .NET.

El .NET Framework está formado por una serie de herramientas y librerías con las que se pueden crear todo tipo de aplicaciones, desde las tradicionales aplicaciones de escritorio, aplicaciones web, desarrollo para móviles, aplicaciones de servidor, etc. [5].

A continuación se detallan las principales bibliotecas del .NET Framework que se utilizan en la arquitectura propuesta:

- ✓ System.Collections, contiene interfaces y clases que definen diversas colecciones de objetos, tales como listas, colas, matrices de bits, tablas hash y diccionarios.
- ✓ System.Data, contiene clases que constituyen la mayoría de la arquitectura ADO. NET[7], permite crear componentes que administran (almacenar, recuperar, manipular y actualizar) eficazmente los datos procedentes de múltiples orígenes, dichos orígenes pueden ser de diferentes tipos, a los cuales se provee un acceso de forma transparente entre los que podemos mencionar se encuentran ODBC, Ole DB, Microsoft SQL Server, SQL Server CE[8], Oracle, MySQL, XML, entre otros.
- ✓ System.IO, contiene tipos que permiten la lectura y escritura de archivos, así como clases para la compresión y descompresión de archivos.
- ✓ System.NET, proporciona una interfaz sencilla para muchos de los protocolos que se utilizan en las redes actuales (Web Server, FTP, DNS, WebService, etc.). Además provee clases que permiten el acceso y actualización de los valores de configuración de la red, así como clases para el envió de correo electrónico a través del protocolo SMTP. También incluye la implementación administrada de la interfaz de Windows Sockets, para controlar el acceso a la red y establecer los canales de comunicación.
- ✓ System.Reflection, contiene clases e interfaces que proveen una vista de los tipos o clases cargadas, métodos y campos, con la capacidad de crear e invocar dinámicamente los tipos. Dicha dinamicidad provee una gran ventaja ya que permite la carga, ejecución e interacción de las bibliotecas diseñadas por los programadores, sin necesidad de realizar cambios en las arquitecturas.
- ✓ System.Security, incluye todas las funciones relacionadas con la seguridad de los sistemas, algoritmos de encriptamiento y control de acceso.
- ✓ System.Xml, proporciona compatibilidad basada en estándares para procesar XML [6].

En segundo lugar se describe el algoritmo de HASH seguro SHA1[9], desarrollado por NIST (National Institute of Standards and Tecnology) junto con NSA (National Security Agency). Es un algoritmo de encriptación que produce una salida de 160 bits(o su equivalente 20 bytes), la máxima cantidad de información o mensaje que puede encriptar es de 2^{64} bits. Es utilizado para encriptar mensajes y en la seguridad de protocolos, con esto lo que se persigue es verificar la integridad de la información que se transmite a través de protocolos de comunicación o de otro medio, haciendo constar que el mensaje recibido es el mismo mensaje que se ha enviado [3].

En tercer lugar tenemos el algoritmo de encriptamiento RSA[10], es un algoritmo muy utilizado de clave pública, donde la seguridad de este algoritmo radica en la factorización del producto de dos números primos enteros muy elevados. Y es un algoritmo que ha resistido todo tipo de ataques cripto analíticos debido a que la factorización de números muy elevados puede llevar muchísimo tiempo [7].

Por último se presenta el concepto de Arquitectura en capas que es la arquitectura en la que está basada la propuesta del diseño, en este tipo de programación el objetivo es separar en varios niveles, capas o componentes las aplicaciones, en donde cada uno agrupa una funcionalidad específica bien definida. La ventaja principal que presenta esta arquitectura es que cada componente es independiente de los demás, y si algún componente cambia o es actualizado no afecta el funcionamiento de los demás componentes.

El más utilizado es en tres capas las cuales son:

[7] ADO.NET es un conjunto de clases que exponen servicios de acceso a datos para el programador de .NET. ADO.NET ofrece abundancia de componentes para la creación de aplicaciones de uso compartido de datos distribuidas. Constituye una parte integral de .NET Framework y proporciona acceso a datos relacionales, XML y de aplicaciones.

[8] SQL Server CE, Base de Datos de dispositivos móviles con sistema operativo Windows.

[9] SHA1, Secure Hash Algorithm versión 1 (1995), Hash valor comúnmente conocido como checksum. Se refiere a una función o método para generar claves o llaves que representen de manera casi unívoca a un documento, registro, archivo, etc. [8].

[10] RSA: (Rivest, Shamir y Adleman) es un sistema criptográfico de clave pública desarrollado en 1977. [7].

- ✓ Presentación, es la interfaz de usuario, captura la información y presenta resultados.
- ✓ Negocio, abstrae toda la lógica del negocio, se definen los procesos, reglas o validaciones que deben cumplirse y algunos cálculos.
- ✓ Datos, es donde se encuentran los datos, por lo general siempre es un gestor de base de datos, el cual es el encargado de almacenar y recuperar información [9].

A continuación se muestra *Diagrama No 1* que ilustra de mejor manera la arquitectura en capas.

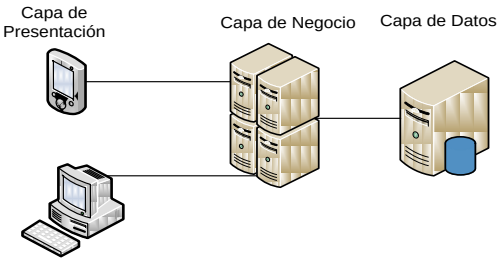

Diagrama No 1 Arquitectura en 3 capas

DISEÑO DE LA ARQUITECTURA

La propuesta de arquitectura que se presenta es el resultado de una recopilación de información, orientada a proponer una infraestructura de comunicación y ejecución de aplicaciones distribuidas.

Se considera necesario un planteamiento basado en componentes independientes, ya que esto nos permite lograr un nivel de reutilización y escalabilidad más amplio. Al mismo tiempo que se habilitan segmentos con los que se podrá interactuar separadamente. Por ejemplo una nueva aplicación podría utilizar los componentes de capa de acceso a datos y de esta forma beneficiarse de sus funcionalidades.

Esto también beneficiará en el trabajo de desarrollo de aplicaciones, en vista que permite separar las tareas de modo que diferentes miembros del equipo de trabajo pueden enfocarse en secciones diferentes de la misma aplicación.

Por otra parte se plantea la inclusión de herramientas tecnológicas de uso extendido, esto permite abrir opciones a diferentes tipos de dispositivos y plataformas se sirvan de los servicios o utilidades existentes.

La siguiente es una lista de los componentes en los que se basa este diseño:

- ✓ Comunicación: compuesto por las funcionalidades que permiten la comunicación a través de la red, manteniendo un enlace constante y válido para el intercambio de información.

La implementación de esta capa estará basada en Sockets de TCP IP por lo que sería posible conectar cualquier tipo de dispositivo PC o móvil siempre y cuando estos sean capaces de establecer enlaces usando dicho tipo de componentes.

El manejo de aspectos de seguridad en cuanto al acceso y transmisión de datos, debe ser parte de los servicios que esta capa ofrece.

Es importante recordar que este es un componente compartido entre ambos lados de la comunicación Cliente y Servidor.

- ✓ Ejecución y Control: es el componente encargado de realizar los llamados a los módulos según sean solicitados desde las aplicaciones Cliente. Los servicios de manejo de resultados y manejo excepciones deben ser provistos para garantizar la máxima disponibilidad del servicio.

El control y manejo de las conexiones así como los módulos a que se desean acceder, deberán ser administrados desde este elemento.

- ✓ Lógica de negocios: concentra la lógica de aplicaciones, componentes o módulos que serán llamados, cargados y utilizados remotamente.

Es necesario conocer la estructura interna de sus interfaces, parámetros y propiedades para que puedan ser llamados desde las aplicaciones Cliente.

- ✓ Acceso a datos: corresponde a los objetos que facilitan la conexión a un gestor de bases de datos. Este administra las conexiones, la ejecución de peticiones y resultados obtenidos de dichos llamados, estos resultados luego serán retornados a las aplicaciones solicitantes.

- ✓ Utilería: reúne un conjunto de funcionalidades reutilizables que sirven de apoyo a los componentes antes mencionados de la arquitectura así como a las aplicaciones que

deseen comunicarse.

El *Apéndice A*, muestra el Diagrama No 30 el cual nos ayudará a obtener una imagen visual de los componentes de la arquitectura previamente mencionados, así como su orden e interacción dentro de la misma.

DISEÑO DE COMPONENTES

A continuación profundizaremos en la estructura interna de cada uno de los componentes de la arquitectura, los detalles deberán permitir una comprensión conceptual de cada elemento así como de los detalles técnicos que ayuden a su implementación.

1. Componente de comunicación

La capa o componente de comunicación se basa en algunos conceptos elementales de la comunicación de aplicaciones de software actuales.

Comenzaremos diciendo que nuestros elementos base serán los Sockets de TCP IP. Donde TCP es un protocolo que contribuye a la comunicación entre dos dispositivos empaquetando y desempaquetando la información que se intercambia. Estos dispositivos PC o móviles deben ser localizables a través de una dirección de red o mejor conocida como dirección IP que los identifique como únicos.

El origen de los Socket tuvo lugar en una variante del sistema operativo Unix conocida como BSD Unix, en la universidad de Berkeley. Implementándose en utilidades de interconectividad de este Sistema Operativo tales como: rlogin, telnet, ftp, entre los más conocidos [4]

La comunicación entre procesos a través de Sockets se basa en la filosofía CLIENTE-SERVIDOR: donde un segmento actuará de Servidor creando un Socket cuyo nombre es conocido por el segmento Cliente, este a su vez podrá "hablar" con el segmento Servidor a través de la conexión con dicho Socket.

Debemos tener claro que la comunicación vía Sockets se basa en el uso de tres recursos:

a) Un protocolo de comunicaciones.
b) Una dirección de Red o dirección IP.
c) Un número de puerto, que identifica a un programa dentro de una computadora.

El proceso de funcionamiento de este mecanismo es mostrado en el Diagrama No 2.

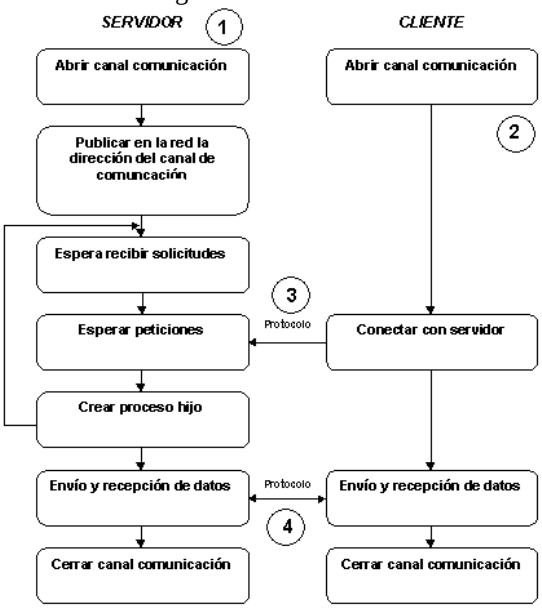

Diagrama No 2 Proceso de comunicación de Sockets

Este mecanismo de comunicación vía Sockets se basa en los siguientes pasos:

1º) El proceso Servidor crea un Socket con nombre y espera las conexiones. Este será accedido desde el Cliente que conoce el nombre de la máquina (hostname) o dirección IP, y el número del puerto utilizados por el Servidor.

2º) El proceso Cliente crea un Socket sin nombre,

3º) El proceso Cliente realiza una petición de conexión al Socket Servidor, si este acepta la conexión, asigna un número de conexión y abre un Socket en un puerto diferente, para que pueda continuar escuchando en el puerto original nuevas peticiones de conexión.

4º) El Cliente realiza el intercambio de información a través de su Socket escribiendo o leyendo mientras sea necesario, al finalizar la conexión debe ser cerrada y los recursos devueltos al sistema operativo.

Es importante aclarar que existen diferentes tipos de Sockets y una de las principales consideraciones a no olvidar es que únicamente podemos comunicar Sockets del mismo tipo. Para esta finalidad la arquitectura se basa en Socket de Flujo (Stream): ya

que proveen un flujo de datos de dos vías, confiable, sin duplicados y además el tamaño de paquete no es una limitante. Este se basa en el protocolo TCP (Transmission Control Protocol).

Para nuestro propósito se ha considerado conveniente crear dos clases que representen ambos segmentos del proceso de comunicación, es decir Cliente y Servidor y encapsularlos en un mismo archivo de biblioteca. Ver Diagrama No 3.

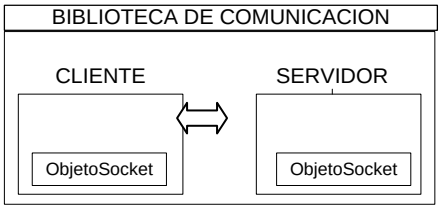

Diagrama No 3 Biblioteca de Comunicación

Como podemos observar ambos elementos comparten la referencia al componente ObjetoSocket, este en realidad no representa una implementación propia de esta arquitectura, sino más bien es el llamado a un componente del Microsoft Framework .NET. La Tabla No. 1 nos detalla la biblioteca propuesta para la utilización de los Sockets.

Tabla No. 1 Descripción de Clase ObjetoSocket

BIBLIOTECA DE SOCKETS	
Versión: Microsoft .NET Framework 3.5	
System.NET.Sockets.Socket	La clase Socket provee un extenso conjunto de métodos y propiedades para comunicarse en red

El Diagrama No 5 muestra la definición de la clase ObjetoSocket, la cual es implementada en los componentes de Cliente y Servidor

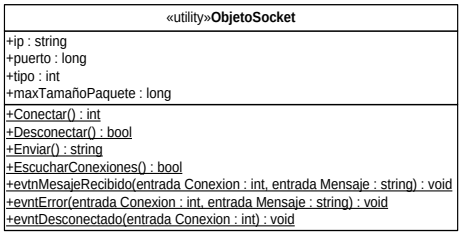

Diagrama No 4 Definición de clase ObjetoSocket

EL CLIENTE

El componente Cliente se encarga de crear su referencia de Socket, y administrar los parámetros y operaciones de conexión al Socket Servidor. Ver Diagrama No 5.

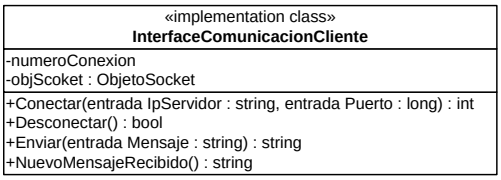

Diagrama No 5 Componente Cliente, definición de clase InterfaceComunicacionCliente.

Donde:

objSocket: corresponde al objeto Socket creado para la comunicación desde el Cliente.

numeroConexion: número identificador correlativo asociado a la conexión.

Conectar: recibe como parámetros la dirección IP y el puerto del Servidor al que se desea conectar.

Desconectar: realiza una solicitud de desconexión al Servidor y apaga el Socket local.

Enviar: envía un mensaje al Servidor.

NuevoMensajeRecibido: lee un mensaje enviado por el Servidor.

En el Apéndice B se nos muestra cuales son las herramientas precisas del System.NET.Sockets.Socket que el Microsoft Framework .NET provee para la implementación de esta capa.

EL SERVIDOR

Por su parte el componente Servidor, abre su Socket para escuchar conexiones que luego encola y administra a través del componente Control y Ejecución el cual será revisado más adelante. Ver Diagrama No 6.

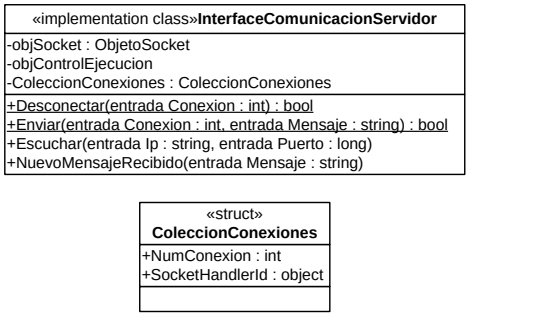

Diagrama No 6 Componente Servidor, definición de clase InterfacecomunicacionServidor

Donde:

objSocket: corresponde al objeto Socket creado para escuchar conexiones.

ColeccionConexiones: propiedad que almacena una colección de conexiones y ayuda crear una matriz de asociación entre el identificador correlativo de las conexiones y el identificador de Socket utilizado internamente por el Microsoft Framework .NET.

Desconectar: ejecuta una solicitud de desconexión de un número de conexión específico. Recordemos que la propiedad ColeccionConexiones nos dice que Socket corresponde cada número de conexión.

Escuchar: operación que indica al Socket del Servidor que debe iniciar la recepción y aceptación de conexiones.

Enviar: envía un mensaje a un número de conexión específico.

NuevoMensajeRecibido: lee un mensaje recibido desde un Cliente.

El Apéndice C lista las referencias y operaciones propuestas para realizar la implementación de este segmento de la arquitectura.

La capa comunicación deberá además proveer otros servicios, tales como Seguridad y Compresión de información.

Para una visualización más clara ver Diagrama No 7.

Diagrama No 7 Arquitectura de Capas de Compresión y Seguridad

Los elementos anteriormente mostrados se deberán incorporar en un orden lógico según el flujo de procesamiento, de acuerdo a nuestro criterio consideramos que la lógica de compresión debe ser implementada sobre la información de usuario que se transmite. Y posteriormente se deberá implementar el aseguramiento de este paquete a través de la capa de Seguridad.

COMPRESION

La utilización de esta funcionalidad es de uso exclusivo de este componente de comunicación, de tal forma que no quedará expuesto para ser accedido por los demás componentes de la arquitectura.

El siguiente diagrama corresponde a la propuesta de diseño de la clase de Compresión, esta puede ser implementada como una clase abstracta en vista que no se considera necesaria la instanciación de un objeto para poder realizar los llamados a sus métodos:

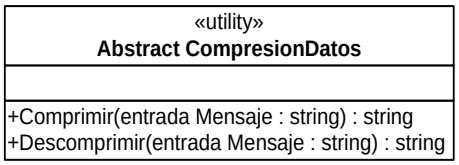

Diagrama No 8 Clase CompresiónDatos

Microsoft Framework .NET implementa herramientas de compresión por medio de las clases System.IO.Compression. Ver Apéndice D para detalles de las clases y métodos útiles para la implementación de este elemento.

SEGURIDAD

Para el control y manejo de la seguridad se propone la utilización de dos mecanismos de aseguramiento:

1. Utilización de un Toquen predefinido y estático empaquetado en un algoritmo SHA1. SHA (Secure Hash Algorithm). SHA es definido como un sistema de funciones hash criptográficas de codificación muy resumida e irreversible. Este toquen será utilizado como una firma digital de los paquetes que esta arquitectura interpreta.

2. Al mismo tiempo se recomienda la utilización de un algoritmo de encriptamiento RSA, basado en el uso de llaves públicas.

La estructura de clase propuesta se presenta en el siguiente Diagrama No 9.

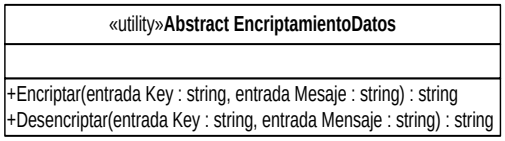

Diagrama No 9 Clase EncriptamientoDatos

Las clases System.Security.Crytography dentro del Microsoft Framework .NET proveen la funcionalidades de encriptamiento para el algoritmo

seleccionado en esta implementación. El Apéndice E nos detalla acerca de esta característica.

EL PROTOCOLO

Finalmente es necesario diseñar el medio que ambos segmentos de la comunicación utilizarán para entenderse entre sí. Para ello debemos definir un mecanismo de mensajería que deberá ser conocido tanto por las aplicaciones Cliente como por el Servidor.

Los protocolos de comunicación constituyen ese conjunto de reglas que gobiernan el formato y el significado de las tramas, paquetes o mensajes que son intercambiados por las entidades que se desean comunicar.

Es importante enfatizar que en este caso dicho protocolo no necesita ser implementado directamente en las aplicaciones Cliente o Servidor. Sino más bien se encuentra abstraído en las capas de Comunicación y Control y Ejecución. Estas son las que deben saber interpretarlo.

Como el propósito de esta arquitectura es la ejecución de aplicaciones remotas, es necesario contar con un esquema que permita la portabilidad de información de las aplicaciones a ejecutar, una sección para el detalle de las operaciones que se desea realizar y finalmente un segmento para la información que se transmitirá en caso de ser necesario. El Diagrama No 10 nos representa la estructura del mensaje a utilizarse.

Aplicación	*Módulo*	*Operación*	*Control*	*Datos*

Diagrama No 10 Diseño del Mensaje del Protocolo

Donde:

Aplicación: provee del nombre de archivo físico de la biblioteca que se desea cargar y ejecutar remotamente.

Módulo: proporciona el nombre del módulo, clase o componente interno de la biblioteca que se necesita instanciar.

Operación: especifica el nombre de la función o método que se ejecutará dentro del módulo.

Control: enumera campos de información de control utilizada por los componentes internos de la arquitectura.

Se propone que el Toquen de firma digital de paquetes sea incluido entre la información de esta estructura.

Datos: este corresponde a un campo abierto por medio del cual las aplicaciones se intercambian datos. Es posible utilizar funcionalidades de serialización del componente de Utilería para la serialización de DataSets, objetos o cualquier otra estructura de datos contenida en el Framework .NET.

El anterior componente de comunicación reúne una serie de funcionalidades clave de acuerdo a los requisitos previamente solicitados, por ejemplo la habilitación de comunicación de aplicaciones a través de red local o externa a la organización a través de internet por medio de TCP IP. El aseguramiento de los datos dentro del canal de comunicación y la utilización de mecanismos de compresión que contribuyen al uso eficiente de los recursos de red.

Para llevar a cabo esta implementación es necesaria la creación de un diccionario de datos que contenga un listado con todas las aplicaciones y funciones, puesto que serán los valores a ser enviados, durante las peticiones.

En el Diagrama No 11 se muestran como están relacionadas las clases del componente de comunicación con el .NET Framework.

2. *COMPONENTE DE CONTROL Y EJECUCIÓN*

El componente de control y ejecución es el encargado de cargar las aplicaciones dinámicamente, permitiendo cargar *n* número de aplicaciones que se encuentran en el componente de lógica de negocios, según sean solicitadas desde los clientes. Entre las ventajas podemos mencionar:

- ✓ Si una aplicación sufre cambios o falla no afectará el funcionamiento del resto de aplicaciones
- ✓ Carga en memoria solo los módulos solicitados por los clientes, haciendo un uso más eficiente de la memoria al no tener que cargar todos los módulos de las aplicaciones.

Otra funcionalidad muy importante de este componente es que cuenta con un control de todos y

cada uno de los clientes que realizan las peticiones.

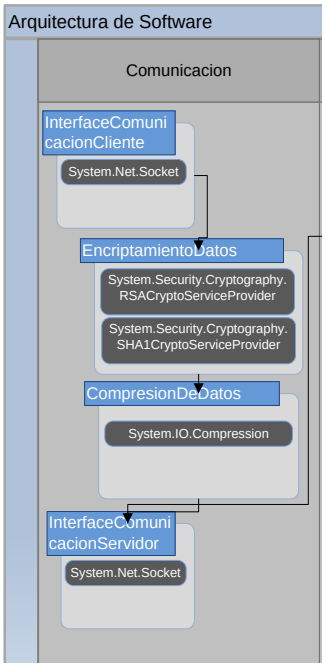

Diagrama No 11 Relación del componente de comunicación con el .NET Framework

INTERACCIÓN CON LOS COMPONENTES

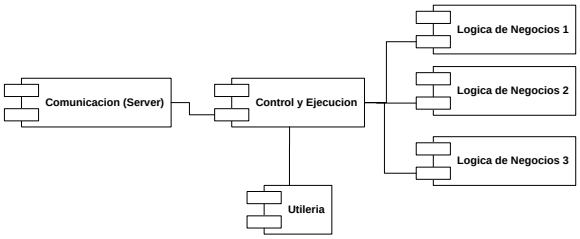

Diagrama No 12 Interacción componente Control y Ejecución con los demás componentes

El componente de Control y Ejecución interactúa directamente con los siguientes componentes:

- ✓ Componente de Comunicación (Servidor), este es el encargado de pasar las peticiones realizadas por los clientes al componente de Control y Ejecución, y cuando ya se tiene un resultado a la petición realizada se encarga de transmitírsela al cliente.
- ✓ Componente de Utilería, este es utilizado cuando se produce algún error, se encarga de generar una entrada de log que describa el error producido y la información necesaria para poder rastrear donde se produjo el error.
- ✓ Componente de Lógica de Negocios, contiene las aplicaciones que serán cargadas desde el componente de Control y Ejecución, procesará las peticiones de los clientes y retornará los resultados a las peticiones.

En el Diagrama No 12 se ilustra de mejor manera la interacción con los diferentes componentes.

OPERACIÓN

La forma en la que opera este componente es la siguiente:

1. El primer paso es cuando recibe la petición desde la capa de comunicación, en donde dicha petición específica que aplicación, módulo, función que se ejecutará y los parámetros necesarios para procesar la petición realizada por el cliente.
2. Una vez recibida la petición se verifica si la aplicación solicitada ya ha sido cargada en memoria, si la aplicación no ha sido cargada se procede a cargarla especificando la ruta de donde se encuentra la aplicación.
3. Cuando ya se ha cargado la aplicación se procede a crear el registro del cliente que está realizando las peticiones, guardando su identificador de conexión, dirección IP desde la que se conecta, así como la fecha y hora de creación del registro del cliente.
4. El siguiente paso a seguir después de haber registrado el cliente que está realizando la petición, es cargarle en memoria el módulo solicitado, al finalizar la carga se asocia el módulo al cliente.
5. Completa la carga del módulo al cliente, se procede a ejecutar la función que se está solicitando.
6. Terminada la ejecución de la petición se retorna el resultado al componente de comunicación para que sea enviado al cliente.

CLASES

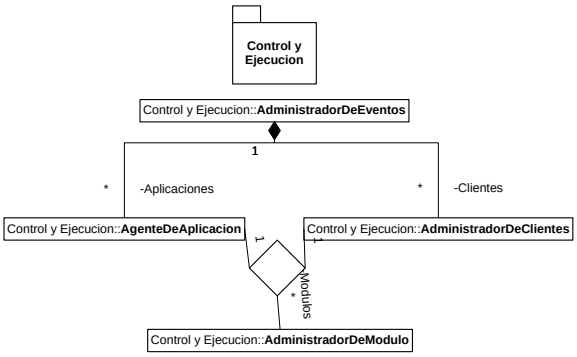

Diagrama No 13 Diagrama de Clases del Componente de Control y Ejecución

En el Diagrama No 13 se muestra el diagrama general de las clases que forman el componente de Control y Ejecución de la arquitectura, el cual está

compuesto por la clase principal de control la cual contiene una colección de clientes que realizan las peticiones, una colección de aplicaciones disponibles para los clientes y las funciones necesarias para cargar las aplicaciones, módulos.

CLASE ADMINISTRADORDEEVENTOS

Control y Ejecucion::**AdministradorDeEventos**
-Clientes : Coleccion -Aplicaciones : Coleccion
+CargarAplicacion() +CargarCliente() +CargarModulo() +Ejecutar() : string +RemoverCliente() : void

Diagrama No 14 Clase Administrador deEventos

La clase de AdministradorDeEventos (ver Diagrama No 14) es la clase principal ya que administra las aplicaciones, así como los clientes que realizan las peticiones.

Propiedades:

- ✓ *Clientes:* Colección que almacena los clientes que han hecho peticiones.
- ✓ *Aplicaciones:* Colección que contiene todas las aplicaciones disponibles para los clientes.

Métodos:

- ✓ *CargarAplicacion():* se verifica si la aplicación ya ha sido cargada en memoria, sino la carga utilizando la siguiente función System.Reflection.Assembly.LoadFrom() del .NET Framework (Ver Apéndice F), a la vez almacena la aplicación en la colección que contiene las aplicaciones.
- ✓ *CargarCliente():* verifica que ya exista un registro para el cliente que realiza las peticiones, sino lo crea y lo almacena en la colección de los clientes.
- ✓ *CargarModulo():* verifica si el módulo ya está cargado en memoria y asociado al cliente, sino lo carga y lo asocia al cliente que realiza la petición a través de la siguiente función System.Reflection.Assembly.CreateInstance() (Ver Apéndice F).
- ✓ *Ejecutar():* ejecuta la petición solicitada por el cliente con la siguiente función System.Type.InvokeMember() del .NET Framework (Ver Apéndice F) y retorna el resultado al cliente.
- ✓ *RemoverCliente():* se ejecuta cuando un cliente se desconecta, se elimina su registro y se libera de memoria los módulos que tiene asociado.

CLASE ADMINISTRARDORDECLIENTES

Control y Ejecucion::**AdministradorDeClientes**
-ClienteID : string -FechaCreacion : DateTime -UltimaActividad : DateTime -TxBytes : double -RxBytes : double -DireccionIP : string -Modulos : Coleccion
+AgregarModulo() : object +RemoverModulo() : void +RemoverModulos() : void

Diagrama No 15 Clase AdministrardorDeClientes

La clase AdministradorDeClientes (ver Diagrama No 15) es la clase que contiene la información de los clientes que están conectados a la aplicación y están activos realizando peticiones.

Propiedades:

- ✓ *ClienteID:* Identificador de la conexión del cliente.
- ✓ *FechaCreacion:* Fecha en la que el cliente realizó una petición por primera vez.
- ✓ *UltimaActividad:* Fecha en la que el cliente realizó la última petición.
- ✓ *TxBytes:* Cantidad de bytes que ha transmitido el cliente a través del componente de Comunicación.
- ✓ *RxBytes:* Cantidad de bytes que ha recibido el cliente a través del componente de Comunicación.
- ✓ *DireccionIP:* Dirección IP desde donde se conecta el cliente.
- ✓ *Modulos:* Colección que almacena los módulos asociados al cliente.

Métodos:

- ✓ *AgregarModulo():* asocia el módulo al cliente y lo almacena en la colección que contiene los módulos.
- ✓ *RemoverModulo():* Elimina el módulo de la colección de los módulos del cliente y lo libera de memoria.
- ✓ *RemoverModulos():* Elimina todos los módulos de la colección de los módulos del cliente y los libera de memoria.

CLASE AGENTEDEAPLICACION

Control y Ejecucion::**AgenteDeAplicacion**
-NombreDeAplicacion : string -Modulos : Coleccion

Diagrama No 16 Clase AgenteDeAplicacion

La clase de AgenteDeAplicacion (ver Diagrama No 16) almacena el objeto que representa a la aplicación que acceden los clientes.

Propiedades:

- ✓ *NombreDeAplicacion:* Identificador de la aplicación.
- ✓ *Modulos:* Colección que almacena los módulos de la aplicación.

CLASE ADMINISTRARDORDEMODULO

Control y Ejecucion::**AdministradorDeModulo**
-NombreDeModulo : string -Funciones : Coleccion

Diagrama No 17 Clase de AdministradorDeModulo

La clase de AdministradorDeModulo (ver Diagrama No 17) almacena el objeto que representa al módulo que acceden los clientes.

Propiedades:

- ✓ *NombreDeModulo:* Identificador del módulo.
- ✓ *Funciones:* Colección que almacena los funciones disponibles en el módulo.

En el Diagrama No 18 se muestra como están relacionadas las clases de los dos componentes descritos hasta el momento, para poder tener una visión global del progreso del diseño de la arquitectura.

3. *UTILERIA*

El componente de Utilería reúne un conjunto de funcionalidades que brindan apoyo al resto de componentes que conforman la arquitectura propuesta, así como a las aplicaciones que se ejecutarán. Entre las funcionalidades que tiene este componente tenemos envío de correo, logueo de errores, serialización de objetos y/o archivos, manipulación de archivos de configuración, llenado de cuadriculas y listas de datos. Entre las ventajas que proporciona este componente tenemos:

- ✓ Reutilización de funciones, lo que significa que no se debe estar creando la misma función en diferentes componentes.
- ✓ El componente puede ser utilizado no solo por la arquitectura propuesta, sino que puede ser utilizado de manera independiente en otras aplicaciones.

INTERACCIÓN CON LOS COMPONENTES

En el Diagrama No 19 se ilustra la interacción con los diferentes componentes.

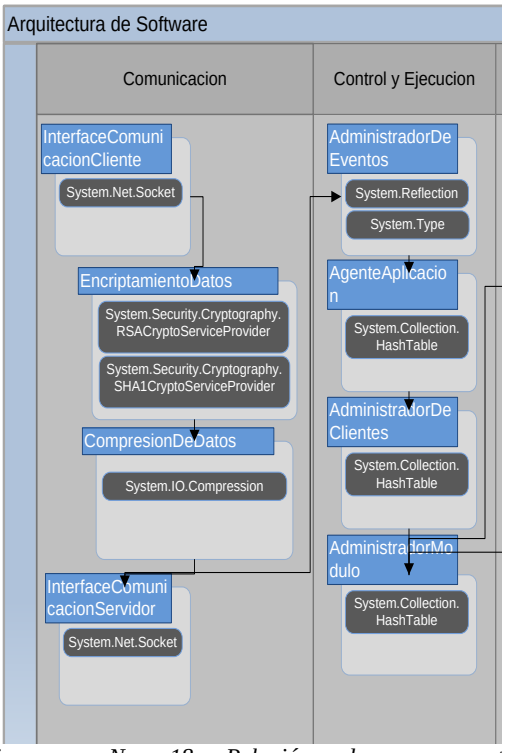

Diagrama No 18 Relación de componentes Comunicación y Control de ejecución con el .NET Framework

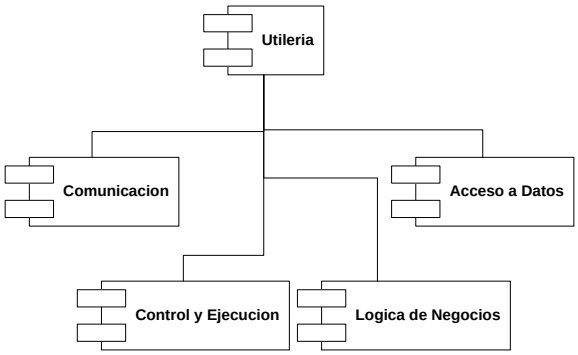

Diagrama No 19 Interacción componente Utilería con los demás componentes

El componente de Utilería prácticamente se comunica o interactúa con todos los componentes que comprende la arquitectura, según se detalla a continuación:

- ✓ Componente de Comunicación, utiliza la función de logueo de errores.

- ✓ Componente de Control y Ejecución, al igual que el anterior utiliza la función de logueo de errores.
- ✓ Componente de Lógica de Negocios, este componente puede hacer uso completo de todas y cada una de las funciones del componente de utilería, todo dependerá si la aplicación requiere o no hacer uso de las funciones.
- ✓ Componente de Acceso a Datos, utiliza la función de logueo de errores.

OPERACIÓN

La forma en la que opera este componente es la siguiente:

1. Lo primero es seleccionar la clase según la funcionalidad que se necesita utilizar, donde la clase puede ser AgenteDeCodificacion, AgenteCorreo, AgenteLlenadoDatos, AgenteLogueo o AgenteDeConfiguración.
2. Una vez seleccionada la clase se invoca la función que se desea utilizar.
3. Por último retorna el resultado de la función.

CLASES

El componente está formado por las siguientes clases: AgenteDeCodificacion, AgenteCorreo, AgenteLlenadoDatos, AgenteLogueo o AgenteDeConfiguracion. A continuación se detalla cada una de las clases.

CLASE DE AGENTEDECODIFICACION

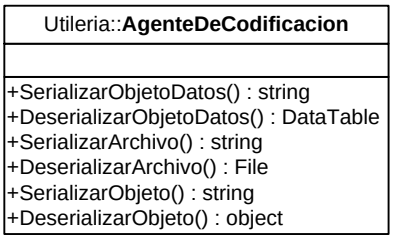

Diagrama No 20 Clase de AgenteDeCodificacion

La clase de AgenteDecodificacion (ver Diagrama No 20) es la que sirve para pasar o codificar objetos a una cadena de caracteres, de forma que dicha cadena de caracteres pueda ser utilizada para diferentes situaciones, tales como almacenarla en la base de datos, para transmitirla a través del componente de comunicación.

Métodos:

SerializarObjetoDatos(): serializa un objeto de datos *System.Data.DataTable* (Ver *Apéndice G)* y lo codifica en una cadena de caracteres.

- ✓ *DeserializarObjetoDatos():* realiza el proceso inverso que el método anterior, este toma la cadena de caracteres que representa el objeto de datos y luego crea dicho objeto.
- ✓ *SerializarArchivo():* convierte un archivo en una cadena de caracteres que es la representación binaria del archivo.
- ✓ *DeserializarArchivo():* realiza el proceso contrario del método anterior.
- ✓ *SerializarObjeto():* serializa objetos y retorna una cadena de caracteres que contiene la representación del objeto en XML.

CLASE DE AGENTELOGUEO

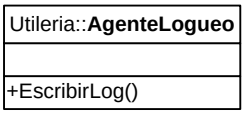

Diagrama No 21 Clase AgenteLogueo

La clase de AgenteLogueo (ver Diagrama No 21) sirve para registrar los errores o fallos que puedan presentarse en la arquitectura.

Métodos:

- ✓ *EscribirLog():* registra los errores ocurridos, recibiendo como parámetro el nombre de la aplicación, nombre del módulo, método de donde se presenta el error, así como una descripción del mensaje de error y lo almacena en un archivo de texto.

CLASE DE AGENTEDECORREO

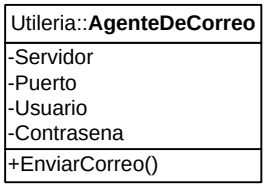

Diagrama No 22 Clase AgenteDeCorreo

La clase de AgenteDeCorreo (ver Diagrama No 22) provee la capacidad de conectarse a un servidor de correo y poder enviar correos electrónicos.

Propiedades:

- ✓ *Servidor:* Dirección IP o URL del servidor de correo.
- ✓ *Puerto:* Puerto del servidor de correo.
- ✓ *Usuario:* El usuario con el que se autentica en el servidor de correo.

- ✓ *Contrasena:* La contraseña del usuario para loguearse al servidor de correo.

Métodos:

- ✓ *EnviarCorreo():* permite el envío de correos electrónicos, recibe como parámetros la dirección hacia donde lo enviará, el tema y el cuerpo del correo que puede ser HTML o texto.

CLASE DE AGENTEDECONFIGURACIÓN

Utileria::**AgenteDeConfiguracion**
-NombreArchivo : string
+CrearArchivo() +AgregarParametro() +ModificarParametro() +LeerParametro() : string

Diagrama No 23 Clase de AgenteDeConfiguración

La clase de AgenteDeConfiguracion (ver Diagrama No 23) se encarga de manipular archivos de configuración.

Propiedades:

- ✓ *NombreArchivo:* representa el nombre del archivo de configuración.

Métodos:

- ✓ *CrearArchivo():* crea el archivo de configuración con formato XML.
- ✓ *AgregarParametro():* agrega un parámetro al archivo de configuración.
- ✓ *ModificarParametro():* modifica un parámetro del archivo de configuración.
- ✓ *LeerParametro():* lee el valor actual del parámetro en el archivo de configuración.

CLASE AGENTELLENADODECONTROL

Utileria::**AgenteLlenadoDeControl**
+LlenarLista() +LlenarCombo() +LlenarGrid()

Diagrama No 24 Clase AgenteLlenadoDeControl

La clase de AgenteLlenadoDeControl (ver Diagrama No 24) incluye la funcionalidad de llenar controles con información proveniente de objetos de datos.

Métodos:

- ✓ *LlenarLista():* llena un control tipo lista (System.Windows.Forms.ListBox) con datos proveniente de un objeto de datos.
- ✓ *LlenarCombo():* llena un control tipo combo (System.Windows.Forms.ComboBox) con datos proveniente de un objeto de datos.
- ✓ *LlenarGrid():* llena un control tipo grid (System.Windows.Forms.DataGridView) con datos proveniente de un objeto de datos.

En el
Apéndice G se muestra la referencia de las funciones del .NET Framework que se utilizan en este componente.

En el Diagrama No 25 se presenta la relación de las clases del componente de Utilería con el .NET Framework.

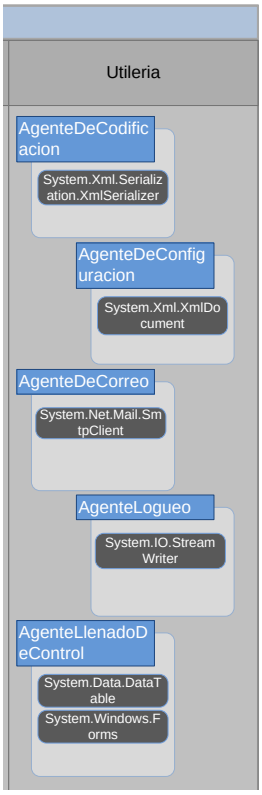

Diagrama No 25 Relación del componente de utilería con el .NET Framework

4. *COMPONENTE DE ACCESO A DATOS*

Este componte es el encargado de facilitar la comunicación con diversos orígenes de datos, como gestores de base de datos, archivos, Web Services o fuentes XML, etc. Este componente de acceso a datos debe contar con una clase por cada tipo de origen de datos al cual se requiera tener acceso; para efecto de "ejemplificación", describiremos la clase que accede a un origen de datos, en este caso será el gestor de

base de datos SQL Server 2008. Ver Diagrama No 26 en el cual se muestra como está estructurada una de las clases que conforma el componente de acceso a datos, así como sus propiedades y métodos disponibles.

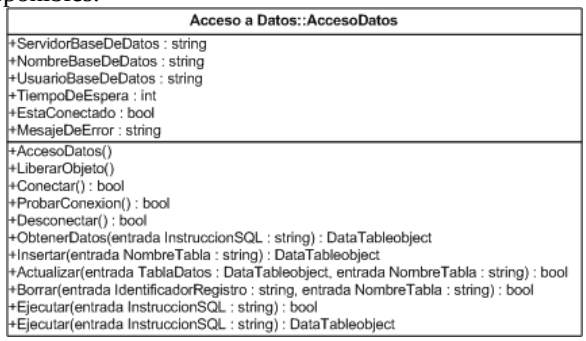

Diagrama No 26 Clase de acceso a datos del componente AccesoDatos

El componente de Acceso a Datos se encarga de administrar las conexiones y todas las peticiones que se realicen al origen de datos, para llevar a cabo esta tarea se apoya en la librería System.Data.SqlClient que ofrece el Framework .NET (*Ver Tabla No 2 Descripción de las clases ADO.Net)*, así mismo se encarga de manejar los resultados obtenidos de las ejecuciones que serán entregados a las aplicaciones que lo han solicitado. Ver Diagrama No 27.

*Tabla No. 2 Descripción de Clase para manejo de datos desde ADO.NET***[10]**

Microsoft .NET Data	
System.Data.SqlClient	El Framework de .NET dispone de este proveedor de datos para SQL Server
System.Data.OleDb	Para orígenes de datos que se exponen mediante OLE DB
System.Data.Odbc	Para orígenes de datos que se exponen mediante ODBC
System.Data.OracleClient.	Para orígenes de datos de Oracle. El proveedor de datos de .NET Framework para Oracle es compatible con la versión 8.1.7 y posteriores del software de cliente de Oracle

Propiedades:

- ✓ *ServidorBaseDeDatos:* especifica el origen de datos
- ✓ *NombreBaseDeDatos:* define la base de datos con la que se va trabajar.
- ✓ *UsuarioBaseDatos:* concatena usuario y contraseña:
- ✓ *TiempoDeEspera:* permite definir el tiempo máximo para esperar respuesta.
- ✓ *EstaConectado:* devuelve un valor verdadero o falso si la conexión está disponible
- ✓ *MensajeError:* devuelve una cadena de texto si hay algún tipo de error.

Métodos:

- ✓ *AccesoDatos():* constructor de la clase
- ✓ *LiberarObjeto():* libera el objeto de memoria.
- ✓ *ProbarConexion():* permite consultar y validad la disponibilidad de la base de datos.
- ✓ *Conectar():* permite abrir la conexión.
- ✓ *Desconectar():*cierra la conexión existente.
- ✓ *ObtenerDatos():* devuelve la información consultada si la hay.
- ✓ *Insertar():* permite crear un nuevo registro.
- ✓ *Actualizar():* actualiza registros.
- ✓ *Borrar():* borra registros.
- ✓ *Ejecutar():* actualiza el origen de datos mediante el uso de comandos directos.

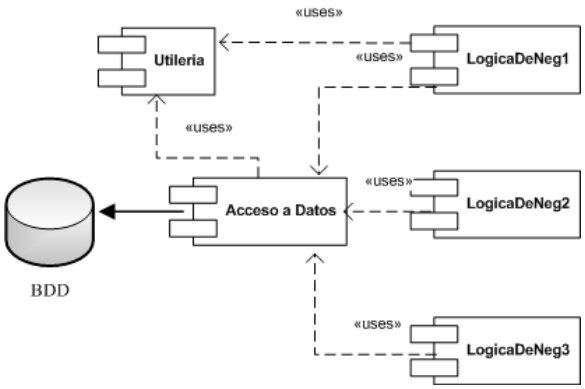

Diagrama No 27 Muestra la interacción entre el componente Acceso a Datos, Utilería y los componentes de Lógica de Negocio.

El componente de Acceso a Datos es el encargado de interactuar entre la base de datos y los componentes que sean necesarios para manejar la lógica del negocio. Cada componente dentro de la lógica de negocio que requiera acceder a la información en el repositorio de datos deberá hacer referencia al componente de Acceso a Datos y puede crear tantas instancias como orígenes de datos requiera acceder, de esta forma dicho componente hará las peticiones o ejecuciones directas al gestor de base de datos y retornará los resultados, para que sean procesados en los componentes que manejan la lógica del negocio.

Inicialmente el componente de lógica de negocio debe crear una instancia del componente Acceso a Datos haciendo referencia a la clase AccesoDatos, configurar las propiedades con los valores que le

permitirán establecer la conexión, como el nombre del servidor o la dirección IP, nombre de la base de datos y las credenciales de usuario para la autenticación que han de configurarse en la propiedad System.Data.SqlClient.SqlConnection.ConnectionString (Ver Apéndice H) del objeto conexión.

Una vez se ha creado la instancia de la clase, este necesita crear una conexión a la base de datos la cual le va a permitir interactuar con el gestor, para esto se ejecuta su método Conectar() que llama a System.Data.SqlClient.SqlConnection (Ver Apéndice H) para crear un objeto de conexión y así iniciar la comunicación invocando su método *Open()* (Ver Apéndice H)

Una vez establecida la conexión, los métodos de la clase ObtenerDatos(), Insertar(), Actualizar(), Ejecutar() del componente de Acceso a Datos que quieren interactuar con la base de datos hacen uso de los métodos, propiedades y objetos de System.Data.SqlClient.SqlDataAdapter (Ver Apéndice H) el cual sirve como un puente que nos permiten recuperar y guardar datos, llenar los objetos DataTable (Ver Apéndice H) , permitiendo manipular los objetos desde y hacia el gestor.[14]

Cuando se hayan realizado todas las operaciones y se desea finalizar la conexión establecida, se hace el llamado del método Desconectar() del componente, el cual internamente hace uso de las propiedades del objeto System.Data.SqlClient.SqlConnection como *State* (Ver Apéndice H) para verificar el estado de la conexión y del método *Close()* (Ver Apéndice H) para cerrar la conexión establecida.

5. *COMPONENTE DE LOGICA DE NEGOCIO*

Esta parte de la arquitectura propuesta puede contener n cantidad de componentes, que internamente encapsulan funcionalidad específica que maneja la lógica del negocio. Cada componente debe estar formado por al menos una clase y exponer aquellos métodos a los cuales se va a permitir el acceso.

Estos componentes implementan las reglas de negocio, aceptan y devuelven estructuras de datos simples y/o complejas que han de ser enviadas a través del componente de ejecución y control hacia los clientes que lo han solicitado. Los componentes de negocio deben exponer funcionalidad de modo que sea independiente de los orígenes de datos y los demás componentes necesarios para realizar lo requerido, deben ser creados de forma que agrupen funcionalidad, significado y lógica.

En el Diagrama No 28 se muestra de forma general como debe estar formado uno de los componentes de lógica de negocio, lo llamaremos LogicaDeNeg1.

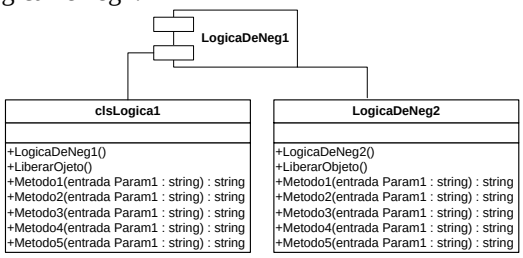

Diagrama No 28 Muestra la forma en la que puede estar estructurado un componente de la lógica de negocio con su respectiva clase.

Esta propuesta de arquitectura es la parte que proporciona la lógica de negocio y que permite expandir su funcionalidad, proporcionando más operaciones que mejoran y encapsulan la lógica existente; de tal forma que si se vuelve necesario agregar nuevas funcionalidades que den soporte a las reglas del negocio; se pueden desarrollar más componentes, agregar clases o bien solo crear nuevos métodos e incorporarlos con facilidad.

Como podemos observar en el Diagrama No 27, cada componente de lógica de negocio puede crear instancias del componente Utilería para hacer uso de sus funciones expuestas, si así lo requiere. De la misma forma puede crear tantas instancias del componente de Acceso a Datos como orígenes de datos se quiera acceder.

Los componentes de lógica de negocios van a ser instanciados desde el componente de Ejecución y Control, como se especificó anteriormente en el apartado 2 en su Diagrama No 12, donde muestra la forma en la que interactúan con los componentes.

Si los componentes de lógica de negocios van a realizar operaciones con los orígenes de datos, deben hacer referencia a la clase System.Data.DataTable (ver Apéndice I) que nos permitirá manipular los resultados obtenidos de las funciones ejecutadas del componentes de acceso a datos.

En el Diagrama No 29 se ilustra una visión completa de la arquitectura y como está vinculado cada uno de sus componentes con el .NET Framework.

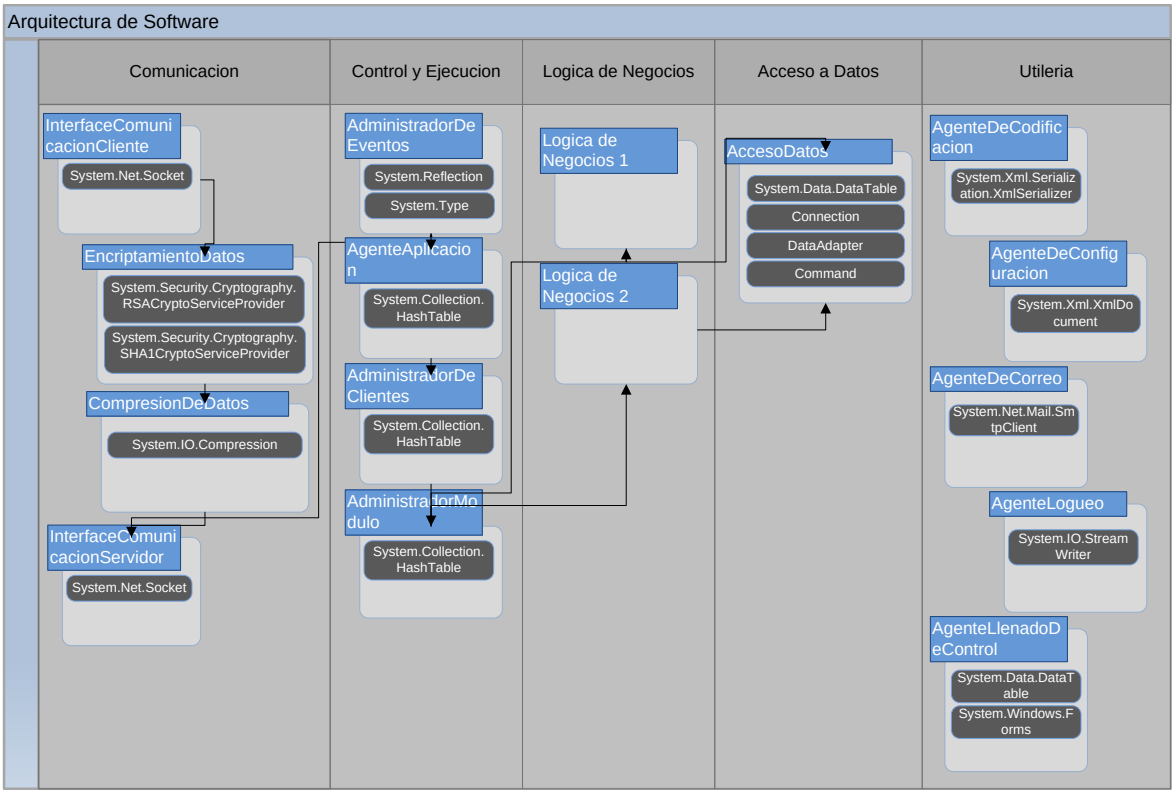

Diagrama No 29 Relación de los componentes de la arquitectura con el .NET Framework

RECOMENDACIONES

- El manejo de excepciones y control de errores, no está contemplado en este diseño, sin embargo es una práctica aconsejable a la hora de la implementación.

- Se podría indagar en la posibilidad de extender la utilización de otros protocolos de comunicación por ejemplo los orientados a no conexión como UDP.

- Se debe tener en cuenta que esta implementación maneja las solicitudes de manera síncrona, por lo cual las aplicaciones Cliente no ejecutará otras operaciones mientras una solicitud se ejecuta. En futuras implementación podría diseñarse e implementarse la versión asíncrona del proceso de comunicación.

- Se sugiere reforzar manejo documental de los puertos que se utilizan para evitar inconvenientes con otras aplicaciones, se recomiendan puertos como: 33010 al 33090.

- El uso y ejecución de aplicaciones remotamente a través de Internet requiere la habilitación y mapeo de puertos en Router o Firewall frontera de nuestra red.

- En vista que IPv4 está llegando a su límite de direcciones de red, se podría realizar una implementación basada en IPv6, obteniendo ventajas como la asignación de direcciones únicas a cada persona o dispositivo del planeta; segundo, minimizar el procesamiento del encabezado de paquetes haciendo más eficiente la comunicación, por mencionar algunas.[11]

- En futuros diseños e implementaciones se recomienda adicionar un componente que se encargue de monitorear y verificar el correcto funcionamiento de la arquitectura y cada uno de

sus componentes.

- El componente de utilería solo contiene un conjunto de clases y funciones básicas para el funcionamiento de la arquitectura propuesta, pero en futuros diseños e implementaciones se pueden adicionar muchas funciones.

- El diseño propuesto está basado en tecnologías Microsoft .NET, pero puede servir como guía para implementar el diseño en otras tecnologías.

- Las librerías utilizadas en el componente de acceso a datos hacen referencia al proveedor de SQL Server, pero pueden agregarse librerías que manejen otros gestores. Se recomienda en futuros diseños, crear métodos o propiedades que permitan definir el gestor de base de datos y sus requerimientos para la conexión y operación.

CONCLUSIONES

La comunicación y ejecución de aplicaciones remotamente es cada vez más un requisito en vez de una funcionalidad extra en la arquitectura de sistemas, la Web es sin duda la herramienta de mayor expansión en ese ámbito. Sin embargo esta no es siempre la más segura y además está sujeta a vulnerabilidades muchas veces lejanas a nuestro control. Una implementación con las características como las que se han presentado puede sin duda ayudarnos a librarnos de muchos de esos inconvenientes.

Sin duda el alcance de esta arquitectura estará por debajo de las capacidades que otras tecnologías mundialmente conocidas podrían proveer. Es por ello que se considera como una herramienta de uso corporativo, es decir para el soporte de operaciones más que para el uso público y abierto a la red.

La arquitectura propuesta puede ser una alternativa para muchas empresas que aun tienen sistemas antiguos y que el presente diseño representa una opción para ellos y así poder ir actualizando sus sistemas de información.

El diseño propuesto es lo suficientemente flexible y escalable que permite la adición de nuevos componentes, siempre y cuando no se afecte la forma en la que interactúan los componentes actuales de la arquitectura.

Se agilizará el desarrollo de aplicaciones ya que permitirá a los desarrolladores enfocarse en la lógica de negocios y no en detalles como la comunicación, seguridad, conexión a orígenes de datos, etc.

Así mismo el encapsulamiento de funcionalidades comunes, permitirá la reutilización del código, lo que también se expresa en reducción de tiempo de desarrollo.

Diseñar una plataforma basada en componentes permite tener control sobre aquellos que se han creado.

Los errores o fallas pueden ser identificados con facilidad, debido a que la modularidad en la que está basada la propuesta, permite aislar las funcionalidades y emitir un diagnóstico certero.

El beneficio de crear componentes que encapsulan los métodos y propiedades de los Assemblies del .NET Framework, es que reduce la necesidad de interactuar directamente con sus clases, en su lugar se exponen métodos que agrupan aquellas operaciones valiosas para el componente que se desarrolla.

APÉNDICES

Apéndice A

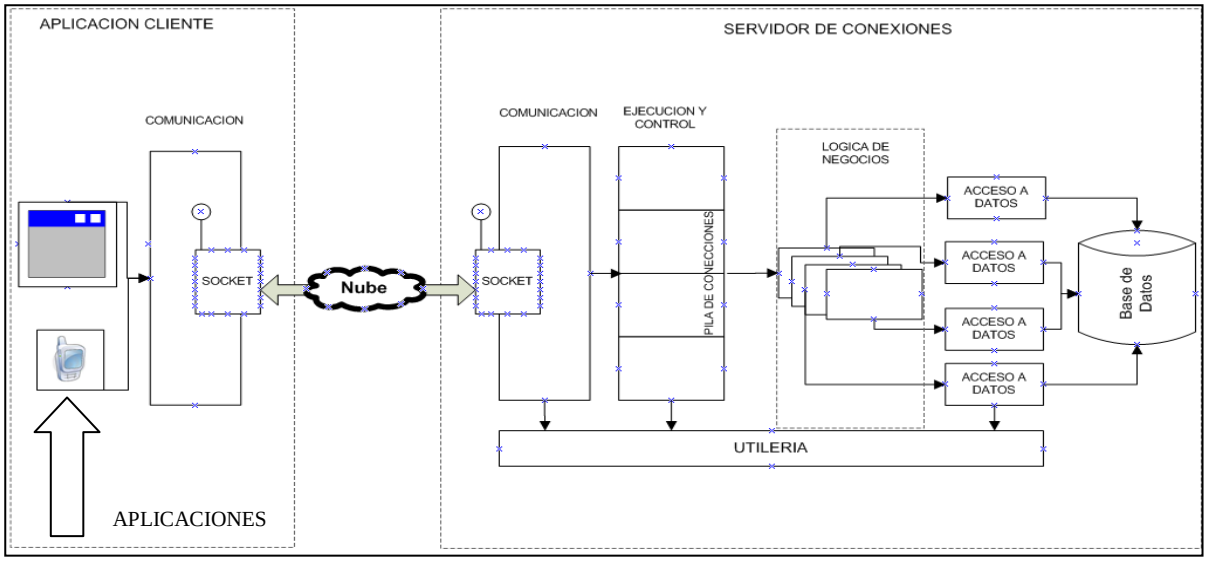

Diagrama No 30 Arquitectura de la solución

El Diagrama No 30 muestra dos tipos de aplicaciones que intercambian información a través de la Capa de Comunicación, esta capa corresponde al segmento más bajo de la arquitectura y habilita la transmisión de datos entre dos puntos Aplicación Cliente (PC o dispositivos móviles) y Servidor de Conexiones.

Una vez los Clientes alcanzan el Servidor de Conexiones, estas son registradas dentro de una Pila de Conexiones por medio de la capa de Ejecución y Control.

La Capa de Ejecución y Control es un módulo de nivel superior que administra y ejecuta los componentes de Lógica de Negocios que han sido solicitados desde la Aplicación Cliente.

En caso que fuera necesario los componentes de Lógica de Negocios pueden tener acceso a una fuente de datos por medio de la capa de Acceso a Datos.

Apéndice B

MÉTODOS DEL OBJETO SOCKET CLIENTE		
Nombre	**Función**	**Implementación**
Conexión. New()	System.NET.Sockets.Socket. *New*(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp)	Constructor de la clase, se establecen los parámetros del Socket y protocolos a utilizar: p.e.: Internetwork y TCP.
Conexión. Conectar()	.*Connect*(*IPEndPoint*)	Establece una conexión con el Socket Servidor. Usando la clase IPEndPoint (hostIP, port). Retorna el Socket Cliente conectado si hubo éxito. IPEndPoint recibe como parámetros en su constructor la IP de servidor y el puerto utilizados para la conexión.
.Desconectar()	*.Disconnect()*	Cierra la conexión del Socket y permite reutilizarlo.
.Conectar()	.*Connected*()	Verifica la existencia de una conexión.
.Enviar()	.*Send*()	Envía un mensaje al modulo de servidor seleccionado.
.NuevoMensaje Recibido()	.*Receive*(mensaje)	Recibe mensaje desde una conexión de Socket.
.Desconectar()	.*Shutdown*()	Deshabilita el Socket.
.Desconectar()	.*Close*()	Libera recursos utilizados por el Socket

Apéndice C

MÉTODOS DEL OBJETO SOCKET SERVIDOR.		
Nombre	**Función**	**Implementación**
Conexión. Escuchar()	System.NET.Sockets.Socket. *New*(AddressFamily.InterNetwork,SocketType.Stream,ProtocolType.Tcp)	Constructor de la clase, se establecen los parámetros del Socket y protocolos a utilizar: p.e.: Internetwork y TCP.
.Escuchar()	.*Bind*(*IPEndPoint*)	Especifica la dirección IP local y el número de puerto. Usando la clase *IPEndPoint*(IP, port)
.Escuchar()	.*Listen*(*MáximoNúmero*)	Método que permite al Servidor iniciar la escucha de conexiones. MáximoNúmero se refiere al número máximo de conexiones que se encolarán.
.Escuchar()	.*Accept*()	Procesa las solicitudes de conexión entrantes en cola y devuelve un Socket para intercambiar datos con el Cliente.
.Enviar()	.*Send*(mensaje)	Envía un mensaje a una conexión Cliente.
.NuevoMensajeRecibido()	.*Receive(mensaje)*	Recibe mensaje desde una conexión de Socket Cliente.
.Destructor()	.*Shutdown*()	Deshabilita el Socket.
.Destructor()	.*Close*()	Libera recursos utilizados por el Socket

Apéndice D

Nombre	Función	Implementación
Compresión. Comprimir()	System.IO.Compression. DeflateStream(Buffer,CompressionMode.Compress)	Clase que implementa un Constructor que recibe como parámetros el mensaje a comprimir y la operación que se desea realizar Compress en este caso.
.Descomprimir()	System.IO.Compression. DeflateStream(Buffer,CompressionMode.Decompress)	Clase que implementa un Constructor que recibe como parámetros el mensaje a comprimir y la operación que se desea realizar Decompress en este caso.

Apéndice E

Nombre	Función	Implementación
Crypto. Encriptar()	System.Security.Cryptography.SHA1CryptoServiceProvider.ComputeHash(mensaje)	Calcula y retorna un arreglo de bytes con el hash del mensaje que recibe como parámetro de entrada. Este hash es irreversible. Solo podremos comparar los hash si conocernos el toquen que lo creó.
Crypto. Encriptar()	System.Security.Cryptography.RSACryptoServiceProvider. ExportParameters (includePrivateParameters)	Método que devuelve la llave pública basada en los parámetros que se utiliza RSA para el encriptamiento.
Crypto. Encriptar()	System.Security.Cryptography.RSACryptoServiceProvider. ImportParameters(Toquen)	Método usado para proveer la llave pública de encriptamiento al objeto RSA.
Crypto. Encriptar()	System.Security.Cryptography.RSACryptoServiceProvider. Encrypt(mensaje)	Encripta el mensaje que recibe como parámetro
Crypto. Desencriptar()	System.Security.Cryptography.RSACryptoServiceProvider. Decrypt(mensaje)	Desencripta el mensaje que recibe como parámetro

Apéndice F

Nombre	Función	Relación con componente	Implementación
Control y Ejecución	System.Reflection.Assembly.LoadFrom(String RutaAplicacion)	Carga en memoria las aplicaciones.	Se proporciona la ruta de donde se encuentra la aplicación, retorna un Assembly o librería (Aplicación).
Control y Ejecución	System.Reflection.Assembly.CreateInstance(String Aplicacion+Modulo)	Carga en memoria los módulos para cada cliente.	Se especifica el nombre de la aplicación a la que pertenece el módulo, así como el nombre del módulo, y se crea una instancia del módulo para cada uno de los clientes que lo utilizarán.
Control y Ejecución	System.Type.InvokeMember(String función,Object Modulo,Object Params)	Ejecuta las funciones solicitadas por los clientes dentro de los módulos.	Ejecuta las peticiones de los clientes, primero se selecciona la aplicación y el módulo al cual pertenece la función, luego se pasan los parámetros necesarios y se retorna el resultado al cliente.
Control y Ejecución	System.Collection.HashTable	Clase que representa las colecciones.	

Apéndice G

Nombre	Función	Relación con componente	Implementación
Utilería	System.Xml.Serialization. XmlSerializer	Utilizada para serializar o deserializar objetos	Clase utilizada para la serialización, recibe como parámetro el tipo de objeto que serializará.
Utilería	System.Xml.Serialization. XmlSerializer.Deserialize()	Deserializa objetos.	Función necesaria para crear los objetos en la función DeserializarObjeto().
Utilería	System.Xml.Serialization. XmlSerializer.Serialize()	Serializa objetos.	Función necesaria para crear serializar los objetos en la función SerializarObjeto().
Utilería	System.IO.StreamWriter	Almacena el registro del log en un archivo.	Esta clase permite manipular archivos de texto en donde se registrarán las entradas de log de errores.
Utilería	System.Xml.XmlDocument	Provee funcionalidad para manipular archivos de configuración en formato XML.	Clase que se utiliza para manipular archivos XML.
Utilería	System.Xml.XmlDocument.Load()	Cargar archivo de configuración.	Carga un archivo XML especificando la dirección y el nombre de donde se encuentra el archivo.
Utilería	System.Xml.XmlDocument.GetElementById()	Leer parámetro del archivo de configuración.	Se especifica el nombre del parámetro a leer, se utiliza en la función LeerParametro() de la clase de configuración.
Utilería	System.Xml.XmlDocument.CreateElement()	Agrega un parámetro al archivo de configuración.	Se especifica el nombre del parámetro a crear y luego se asigna el valor del parámetro, se utiliza en la función AgregarParametro() de la clase de configuración.
Utilería	System.Xml.XmlDocument.Save()	Guarda el archivo de configuración.	Guarda un archivo XML especificando la dirección y el nombre donde se guardará el archivo.
Utilería	System.Net.Mail.SmtpClient	Clase necesaria para el envío de correo electrónico	Utilizada en la clase de correo, se necesita especificar el nombre del servidor, el puerto del servidor de correo y las credenciales para loguearse al servidor de correo.
Utilería	System.Net.Mail.SmtpClient.Send()	Envía correo electronic.	Se utiliza en la función de EnviarCorreo() de la clase de correo.
Utilería	System.Data.DataTable	Objeto de datos	Objetos de datos que se utiliza en la arquitectura propuesta

Apéndice G. Continuación.

Nombre	Función	Relación con componente	Implementación
Utilería	System.Windows.Forms.ListBox	Lista de datos a llenar con un objeto de datos.	Utilizada en la función LlenarLista() de la clase de data, a partir de un objeto de datos.
Utilería	System.Windows.Forms.ComboBox	Lista desplegable a llenar con un objeto de datos.	Utilizada en la función LlenarCombo() de la clase de data, a partir de un objeto de datos.
Utilería	System.Windows.Forms.DataGridView	Grid a llenar con un objeto de datos.	Utilizada en la función LlenarGrid() de la clase de data, a partir de un objeto de datos.

Apéndice H

System.Data			
Nombre	**Función**	**Relación con componente**	**Implementación**
Acceso a Datos	DataTable	Permite manipular tablas del origen de datos en memoria.	Objeto utilizado en los métodos ObtenerDatos() para retornar registros, Insertar() para crear registros nuevos y devolverlos, Actualizar() para actualizar datos.
Acceso a Datos	DataRow	Permite manipular registros.	Utilizado cada vez que se requiere manipular un registro de un DataTable.
Acceso a Datos	SqlDataClient.SqlConnection	Establecer conexiones al origen de datos.	Se crea un objeto que manejara la conexión dentro del método Conectar()
Acceso a Datos	SqlDataClient.SqlConnection.ConnectionString	Define origen de datos	Esta propiedad se configura con las credenciales de autenticación requeridas para el origen de datos dentro del método Conectar().
Acceso a Datos	SqlDataClient.SqlConnection.Open()	Abre la conexión con el origen de datos.	Una vez se ha definido ConnectionString, se procede a abrir la conexión.
Acceso a Datos	SqlDataClient.SqlConnection.State	Permite verificar el estado de la conexión	Cada vez que se requiere interactuar con el origen de datos, se verifica esta propiedad para ver si la conexión aun está disponible.
Acceso a Datos	SqlConnection.Close()	Cierra la conexión con el origen de datos	Cuando la conexión ya no va a ser utilizada, se hace el llamado a esta función para finalizar la conexión dentro del método Desconectar().

Apéndice H. Continuación.

System.Data			
Nombre	**Función**	**Relación con componente**	**Implementación**
Acceso a Datos	**SqlClient.SqlDataAdapter**	**Sirve como puente entre el gestor de base datos y los objetos del Framework .NET que permiten manipular los datos.**	**Es necesario crear un objeto de este tipo cada vez que se requiere consultar, insertar o actualizar información del gestor de base de datos en este caso a través de un DataTable.**
Acceso a Datos	**SqlClient.SqlDataAdapter.InserCommand**	**Insertar datos**	**Esta sentencia es utilizada por el método Insertar(), cada vez que requiere agregar datos en el origen de datos.**
Acceso a Datos	**SqlClient.SqlDataAdapter.UpdateCommand**	**Actualizar datos**	**Esta sentencia es utilizada por el método Actualizar(), cada vez que requiere actualizar datos en el origen de datos.**
Acceso a Datos	**SqlClient.SqlDataAdapter.DeleteCommand**	**Elimina datos**	**Esta sentencia es utilizada por el método Borrar(), cada vez que requiere borrar datos en el origen de datos.**
Acceso a Datos	**SqlClient.SqlCommandBuilder**	**Utilizado para actualización de datos.**	**Utilizado para generar automáticamente las instrucciones de insert, update y delete en base a la estructura de la tabla que se quiera modificar.**
Acceso a Datos	**SqlCliente.SqlCommand**	**Permite definir o capturar instrucciones T-SQL para manipular los datos**	**Utilizado cada vez que se quiere hacer referencia a los comandos generados por SqlCommandBuilder dentro de los métodos Insertar(), Actualizar(), Borrar()**

Apéndice I

System.Data			
Nombre	**Función**	**Relación con componente**	**Implementación**
Lógica de Negocio	DataTable	Objeto de datos	Utilizado cada vez que se van a manipular los datos devueltos por el componente de Acceso a datos.

REFERENCIAS

1. Microsoft. (2007). MSDN. Socket Recuperado el 25 de 8 de 2011, de http://msdn.microsoft.com/es-es/library/system.net.Sockets.Socket%28v=VS.90%29.aspx

2. W3Schools. (s.f.). Transmission Control Protocol / Internet Protocol. Recuperado el 18 de Agosto de 2011 de Agosto de 2011, de http://www.w3schools.com/tcpip/tcpip_intro.asp

3. W3C. SHA1 Secure Hash Algorithm. Recuperado el 24 de 8 de 2011, de http://www.w3.org/PICS/DSig/SHA1_1_0.html

4. Zacatepec, I. T. (2010). Comunicación Cliente Servidor Sockets. Recuperado el 20 de 08 de 2011, de http://www.mitecnologico.com/Main/ComunicacionClienteServidorSockets

5. Microsoft. (s.f.). Recuperado el 23 de Agosto de 2011, de Centro de desarrollo de .NET Framework: http://msdn.microsoft.com/es-es/netframework/default

6. Microsoft. (s.f.). Recuperado el 23 de Agosto de 2011, de Referencia de la biblioteca de clases de .NET Framework: http://msdn.microsoft.com/es-es/library/d11h6832%28v=VS.90%29.aspx

7. RSA. (s.f.). Recuperado el 24 de Agosto de 2011, de What is the RSA cryptosystem?. http://www.rsa.com/rsalabs/node.asp?id=2214

8. W3C. (s.f.). *SHA1 version 1.0*. Recuperado el 20 de Agosto de 2011, de World Wide Web Consortium (W3C): http://www.w3.org/PICS/DSig/SHA1_1_0.html

9. MSDN. (s.f.). *Layered Application Guidelines*. Recuperado el 23 de Agosto de 2011, de: http://msdn.microsoft.com/en-us/library/ee658109.aspx

10. Microsoft. (s.f). Recuperado el 25 de Agosto de 2011, System.Data (Espacio de nombres): http://msdn.microsoft.com/es-es/library/system.data(v=vs.80).aspx

11. IPv6 México. (2011). Recuperado el 30 de Agosto 2011. http://www.ipv6.unam.mx/

12. W3C 2011 Guía Breve de Tecnologías XML. Recuperado el 30 de Agosto 2011. De:http://www.w3c.es/divulgacion/guiasbreves/tecnologiasxml

13. W3School (2011) Web Services Turorial. Recuperado el 30 de Agosto 2011. de http://www.w3schools.com/webservices/default.asp

14. Microsoft (2011) SQL Client. Recuperado el 25 de Agosto 2011, de: http://msdn.microsoft.com/es-es/library/system.data.sqlclient(VS.80).aspx

15. Object Management Group, Inc (2011) Unified Modeling Language. Recuperado el 21 de Agosto 2011, de http://www.uml.org/